03

# Extendiendo una excepción

## Transcripción

[00:00] ¿Qué tal? Bienvenidos nuevamente? Continuando el video anterior, ¿qué es lo que había pasado? En nuestra clase Flujo no estamos, no estamos consiguiendo compilar esta clase, porque el método 1 nos dice que hay una excepción que no está siendo tratada correctamente. Entonces, vamos a analizar aquí.

[00:18] Método 1 está declarando que en la firma del método, throws MiException, entonces esto va a obligar al compilador a verificar que estamos haciendo tratamiento de MiException. ¿Eso por qué? Tenemos que recordar que yo aquí estoy extendiendo de Exception, entonces la excepción es verificada, checked. Y por lo tanto, yo sí o sí preciso hacer un tratamiento de este tipo de error.

[00:52] Vimos que ese caso se dio aquí a nivel de cuenta y yo conseguí y tratar como una estructura try/catch. Ahora, quizás en algunos casos puedan pensar: "bueno, pero si va a ser tan trabajoso tratar cada excepción ¿por qué no extiendo directamente de RuntimeException, guardo y el código ya vuelve a compilar tranquilamente?"

[01:15] Y no es una solución válida. ¿Por qué? Porque tenemos que recordar que si nosotros necesitamos o queremos asegurarnos que MiException siempre va a ser tratado obligatoriamente por el programador nosotros nos apoyamos en el compilador para que él verifique siempre que esta excepción de aquí MiException, está siendo tratado.

[01:41] Entonces volvemos aquí nuestra clase Flujo y ya hemos visto que método 1 no compila. Vamos a aplicar la misma estrategia que aplicábamos con la clase TestCuentaException. ¿Qué vamos a hacer? Vamos a decirle que añada un try/catch y listo, básicamente eso sería todo. ¿Pero qué es lo que yo les quiero mostrar aquí?

[02:05] Si recordamos las clases anteriores, ¿qué sucedería si después de método 2? O mejor dicho, aquí, en este punto de aquí, Inicio método 2. Si yo hiciera esto de aquí y a va a ser igual a 50 entre 0. ¿Qué les parece? Esto va a dar un error conocido. Esto de aquí va a lanzar un ArithmeticException. ¿Será que yo puedo atrapar ese ArithmeticException, aparte de MiException que yo estoy obligado a atrapar?

[02:51] Porque si se dan cuenta, aquí él va a dar una excepción y ya lo hemos visto, lo hemos comprobado. Pero el compilador no importa si es que estamos o no haciendo un tratamiento de esa excepción. Yo sé que va a dar esa excepción porque ya nos ha pasado, lo hemos visto. Pero en este caso, bueno, el compilador no va a hacer nada. ¿Qué podría hacer yo? Ya lo hemos visto.

[03:15] Yo añado un pipe ¿y pongo qué cosa? ArithmeticException, ¿y con eso yo de qué me aseguró? Entonces estoy atrapando MiException y estoy atrapando ArithmeticException. ¿Ahora qué tal si yo aquí también hago algo como esto? Cuenta c igual null, y aquí llamo al método deposita. ¿Qué les parece? Esto de aquí ya lo hemos visto también. ¿Qué tipo de excepción va a dar? NullPointerException.

[03:48] Volviendo al Código. ¿Estoy atrapando NullPointerException en algún lado? No. Pero el código compila, y yo sé que él va a dar NullPointerException porque ya lo hemos provocado antes. Ahora solo estamos trayendo errores que ya habíamos cometido para forzar a que ocurran. ¿Y qué podría yo hacer aquí en ese caso?

[04:14] Perfecto, entonces sí también sé que va a ocurrir NullPointerException, lo agrego aquí con NullPointerException. Perfecto. Entonces yo aquí ya tengo las tres excepciones atrapadas y aquí él está compilando. Porque bueno, claro, en el caso del NullPointerException él no puede ser lanzado. Yo no puedo hacer catch de él porque no extiende directamente de Throwable.

[04:46] Como yo no lo estoy lanzando directamente, no lo puedo atrapar aquí a ese nivel, pero entonces vamos a quedarnos únicamente con estos dos, MiException y ArithmeticException. Y si volvemos a la estructura, a la jerarquía de las excepciones, de cómo es que están agrupadas, vamos a ver que todas extienden de Exception tanto Runtime que es el caso de Arithmetic y NullPointerException como MiException.

[05:13] De alguna forma todas llegan a excepción, ¿entonces será posible que yo consiga, digamos atrapar cualquier tipo de excepción en una sola sin necesidad de escribir tantas excepciones aquí? ¿Ustedes que opinan? La respuesta es sí, y si ustedes prestaron mucha atención a las clases de polimorfismo y herencia, van a saber muy rápido por qué.

[05:45] Es que si yo aquí digo abierta y directamente Exception, él va a hacer catch de cualquier excepción que ocurra en mi código, cualquier clase que extienda de Exception, que como vimos son todas las Exception que tenemos disponibles, tanto checked como unchecked. ¿Cómo vamos a asegurarnos de eso? Vamos a probar aquí.

[06:13] Y vemos aquí que él lanzó el ArithmeticException, pero me infinalizó. ¿Por qué? Porque yo hice tratamiento. Hice tratamiento de Exception. Cualquier cosa que suceda aquí, yo me hago cargo, con Exception. Entonces, borrando esto y lanzando directamente MiException, puedo lanzar aquí, voy a decirle que sí lo quiero guardar, y MiException fue lanzada pero también fue atrapada de manera genérica con Exception.

[06:53] Entonces esta es una forma más de digamos atrapar cualquier tipo de excepción. Pueden haber casos en los que sea necesario, hay casos en los que es mucho mejor dejar clara y explícitamente que yo estoy atrapando MiException. ¿Por qué? Porque van a haber veces que yo quiero dejar bien en claro a nivel del código qué tipo de error estoy tratando aquí, para el siguiente programador que venga a hacer un mantenimiento en este código.

[07:17] Entonces, depende mucho del caso, pero la idea aquí es mostrarles que también usando los conceptos de herencia y polimorfismo, también podemos aplicarlo para el manejo de excepciones. Nos vemos en el siguiente video.